Nintendo GameCube™ THP Library

Version 1.0a

Table of Contents

DOL-06-0183-001-A1
Released: 3/28/02

List of Tables

List of Code Examples

List of Figures

List of Formulas

1. Introduction

1.1 Overview

The THP library is designed for the playback of movies on the Nintendo GameCube™. By using the THP library, movie data comprising interleaved video data and audio data (henceforth called THP movie data) can be played on the Nintendo GameCube.

The format of the video data handled by the THP library (henceforth called THP video data) is customized for rapid decoding by the Nintendo GameCube. The THP library has been optimized to a high extent for decoding of this THP video data.

Note: For increasing speed, the THP library uses certain functions unique to the Gekko CPU (i.e., locked cache as well as GQR).

Because THP movie data can be decoded so rapidly, it is extremely well suited for applications inside the game. For example, a movie displayed to a screen size of 640 x 480 pixels at a frame rate of 60 frames per second that has been compressed to around 1bit per pixel, requires around a 60% load on the CPU when it is being decoded. If the frame rate is 30 frames per second, the compression ratio can be lowered to 5 or 6 bits per pixel (thereby raising picture quality).

Note: The actual compression ratio is influenced by limits on the speed at which data is read from the optical disc. The slowest optical disc read speed is 2Mbytes per second. Accordingly, for a movie displayed at 30 frames per second, you want to compress the data to 68.3K per frame or less.

By utilizing high-speed THP movie data, you can decode and display a movie in real-time while rendering objects in front of the movie.

THP video data is created from the conversion of normal JPEG data. The JPEG data is converted into THP video very quickly and with no loss in picture quality from the original JPEG data.

Through the use of THP movie data, the developer can estimate how much time is needed to decode movie data and what kind of picture quality will be realized on the Nintendo GameCube when the data is actually decoded.

The THP movie data's audio data (henceforth called the THP audio data) is compressed in the same format as the Nintendo GameCube audio system's DSP ADPCM. This THP audio data is appropriately processed in order to synchronize the video data and the audio data as required for the movie.

In summary, use of the THP library enables developers to playback high picture-quality and sound-quality movies with only a minimum burden on the CPU.

DOL-06-0183-001-A1
Released: 3/28/02

1.2 Sample program

Every frame of THP video data is compressed independently of every other frame. Because of this, movie playback is easy to realize with the THP library compared to other movie libraries, which require that information from prior and subsequent frames be referenced when decoding the present frame.

However, a slightly complicated procedure is required, in order to achieve appropriate synchronization of the video data and the audio data during movie playback.

To avoid complexity, two movie players have been prepared that hide this procedure of the THP library, making the playback of movies even simpler.

The first movie player is an extremely simple player designed to provide a basic understanding of how to use the THP library.

The second movie player is an advanced player that assumes use in applications. This advanced player has been created with flexibility to meet the various needs of the game with regard to movie playback.

The THP library was intentionally created so that movies could be actively utilized inside even ordinary game scenes. However, the game system can have a large effect on the movie player when it is used in this way. Thus, even though the THP movie player library is easy to use, the developer should modify the configuration of the player to suit the environment.

Source code has been distributed along with the movie players. Feel free to modify this source to suit the THP movie data playback environment.

1.3 The organization of this document

This document is organized as follows:

Section 2 explains the minimal procedure required for playing THP movie data on the Nintendo GameCube.

Section 3 explains the API functions that have been prepared for the procedure described in Section 2.

Section 4 explains the API functions that have been prepared for the more advanced playback of THP movie data on the Nintendo GameCube.

Section 5 explains the low level API functions that realize playback of THP movie data.

Section 6 explains how to create THP movie data.

Section 7 explains cautions regarding use of the THP library.

2. Using the Simple Player to Playback THP Movie Data

The THP simple player (henceforth called the Simple Player) is a THP movie player that provides the minimal set of functions needed for playing THP movie data on the Nintendo GameCube.

This section explains how to use the Simple Player.

2.1 Source files

The Simple Player is supplied as the sample program `THPSimple` in the THP library (build/demos/thpdemo). The Simple Player is comprised of the following source files:

- `src/THPSimple/THPSimple.c`
  Describes the various API functions of the Simple Player. The player calls the THP library's low level API function THPVideoDecode when decoding THP video data and the low level API function THPAudioDecode when decoding THP audio data.
- `src/THPSimple/THPDraw.c`
  The THPVideoDecode function, which is called by the Simple Player when decoding THP video data, outputs the decoded data in YUV texture format. This file describes the functions for drawing this data to the EFB via the graphics processor.
- `src/THPSimple/main.c`
  This is a simple sample of THP movie data for playback using the Simple Player.
- `include/THPSimple.h`
  Contains the definitions for the various API functions of the Simple Player.
- `include/THPDraw.h`
  Contains the definitions for the functions used for drawing the decoded YUV texture format data to the EFB via the graphics processor.

2.2 How to use the Simple Player

Below is a simple program using the Simple Player for playback of THP movie data.

This program is described in build/demos/thpdemo/src/THPSimnple/main.c.

```
1:  #include <demo.h>
2:  #include <stdlib.h>
3:  #include <string.h>
4:  #include "THPSimple.h"
5:  #include "axseq.h"
6:
7:  void main(void)
8:  {
9:      u32             size, x, y, count;
10:     s32             frame, start, vol;
11:     u16             buttonDown, button;
12:     u8              *buffer;
13:     GXRenderModeObj *rmode;
14:     THPVideoInfo    videoInfo;
15:
16:     DEMOInit(&GXNtsc480Int);
17:
18:     AIInit(NULL);
19:
20:     AXSeqSetup();
21:
22:     THPSimpleInit(WITH_AX);
23:
24:     GXSetDispCopyGamma(GX_GM_1_0);
25:
26:     // Open THP file
27:     if (THPSimpleOpen("thpdemo/fish.thp") == FALSE)
28:     {
```

```
29:         OSHalt("THPSimpleOpen fail");
30:     }
31:
32:     THPSimpleGetVideoInfo(&videoInfo);
33:
34:     rmode = DEMOGetRenderModeObj();
35:
36:     x = (rmode->fbWidth   - videoInfo.xSize) / 2;
37:     y = (rmode->efbHeight - videoInfo.ySize) / 2;
38:
39:     // Reserve work area
40:     size = THPSimpleCalcNeedMemory();
41:
42:     buffer = (u8 *)OSAlloc(size);
43:
44:     if (!buffer)
45:     {
46:         OSHalt("Can't allocate the memory\n");
47:     }
48:
49:     THPSimpleSetBuffer(buffer);
50:
51:     OSReport("\n########################################\n");
52:     OSReport("Push A button     : restart the movie\n");
53:     OSReport("Push B button     : play/stop midi file using AX\n");
54:     OSReport("Push Start button : application end\n");
55:     OSReport("Pad up            : volume up\n");
56:     OSReport("Pad down          : volume down\n");
57:     OSReport("########################################\n");
58:
59: RESTART:
60:     // Pre-read for playback
61:     if (THPSimplePreLoad(LOOP) == FALSE)
62:     {
63:         OSHalt("THPSimplePreLoad fail");
64:     }
65:
66:     start = 1;
67:
68:     count = VIGetRetraceCount();
69:
70:     while(1)
71:     {
72:         DEMOPadRead();
73:
74:         buttonDown = DEMOPadGetButtonDown(PAD_CHAN0);
75:         button     = DEMOPadGetButton(PAD_CHAN0);
76:
77:         DEMOBeforeRender();
78:
79:         // Decode 1 frame of THP data
80:         if (THPSimpleDecode() == VIDEO_DECODE_FAIL)
81:         {
82:             OSHalt("Fail to decode video data");
83:         }
84:
85:         // Draw decoded THP video data
86:         frame = THPSimpleDrawCurrentFrame(rmode, x, y, videoInfo.xSize, videoInfo.ySize);
87:
88:         while (VIGetRetraceCount() < count + 1)
89:         {
90:             VIWaitForRetrace();
91:         }
92:
93:         DEMODoneRender();
94:
95:         count = VIGetRetraceCount();
96:
97:         if (start)
98:         {
99:             // Permit audio output
100:            THPSimpleAudioStart();
101:            start = 0;
102:        }
103:
104:        if (buttonDown & PAD_BUTTON_A)
105:        {
106:            // Stop playback and restart
107:            THPSimpleAudioStop();
108:            THPSimpleLoadStop();
109:            goto RESTART;
```

```
110:        }
111:
112:        if (buttonDown & PAD_BUTTON_START)
113:        {
114:            // End playback and exit main loop
115:            THPSimpleAudioStop();
116:            THPSimpleLoadStop();
117:            THPSimpleClose();
118:
119:            OSFree(buffer);
120:
121:            break;
122:        }
123:
124:        if (buttonDown & PAD_BUTTON_B)
125:        {
126:            if (GetSeqState())
127:            {
128:                SeqStop();
129:            }
130:            else
131:            {
132:                SeqPlay();
133:            }
134:        }
135:
136:        if (button & PAD_BUTTON_UP)
137:        {
138:            vol = THPSimpleGetVolume() + 1;
139:            if (vol > 127)
140:            {
141:                vol = 127;
142:            }
143:            THPSimpleSetVolume(vol, 0);
144:        }
145:
146:        if (button & PAD_BUTTON_DOWN)
147:        {
148:            vol = THPSimpleGetVolume() - 1;
149:            if (vol < 0)
150:            {
151:                vol = 0;
152:            }
153:            THPSimpleSetVolume(vol, 0);
154:        }
155:    }
156:
157:    THPSimpleQuit();
158:
159:    VIWaitForRetrace();
160:
161:    OSHalt("application end\n");
162:
163:    return;
164:   }
```

Code 1 Example of use of Simple Player (main.c)

Line 14:

The THPVideoInfo structure is declared. This structure maintains the vertical and horizontal size of the THP video data. The members of this structure are set by the `THPSimpleGetVideoInfo` function (see line 32).

Line 18:

Before initializing one of the Nintendo GameCube audio libraries, AX library, the `AIInit` function is called to initialize the audio interface.

Line 20:

The `AXSeqSetup` function initializes AX internally, and makes preparations for the use of the AX sequencer.

DOL-06-0183-001-A1
Released: 3/28/02

Line 22:

The `THPSimpleInit` function must be called first, before using any of the Simple Player API functions. In addition to initializing the Simple Player's control structure (the THPSimple structure), the `THPSimpleInit` function enables locked cache and calls the THP library's low level API function `THPInit`. It also registers a callback function in the Nintendo GameCube audio interface for the playback of THP audio data. When used with AX and MusyX simultaneously, initialization functions for each audio library (`AXInit, sndInit`) need to be called before `THPSimpleInit` is called. When used with AX simultaneously, `THPSimpleInit` needs to be called while sound output of AX is set to produce no sound. This sample program specifies **WITH_AX** for the argument of `THPSimpleInit` for simultaneous use of AX.

Line 27:

This opens the THP movie data ("fish.thp") specified by the argument. The `THPSimpleOpen` function calls the `DVDOpen` function to open the THP movie data specified by the argument, and calls the `DVDRead` function to read the header portion of that data. After that, the `THPSimpleOpen` function analyzes the header portion to check that the data specified by the argument is valid THP movie data.

Line 32:

This gets information regarding THP video data from the header portion of the THP movie data that was loaded into main memory in line 27, and stores it in the `THPVideoInfo` structure specified by the argument. The information acquired here is utilized by the application (in this case, the sample program described in `main.c`).

Lines 36, 37:

Requests the draw-location (upper left corner coordinates) of the decoded data so the THP video data will be drawn centered in the screen when the THP movie data is played back.

Line 40:

Requests the size of the work region to be used by the Simple Player. The `THPSimpleCalcNeedMemory` function returns a value that is the sum of the size of the buffer used when the THP movie data is read from the optical disc, the size of the YUV texture buffer used when the THP library's low level API function decodes the THP video data, the size of the buffer used by the THP audio data, and the size of the work region used by the THP library's low level API functions. The formula used for this calculation is given below:

```
 // Size of buffer for reading the THP movie data
size  = OSRoundUp32B (Maximum data size of THP frame data) * READ_BUFFER_NUM;

 // Size of Y texture buffer
 + OSRoundUp32B (THP video data horizontal size x vertical size)

 // Size of U texture buffer
 + OSRoundUp32B (THP video data horizontal size x vertical size/4)

 // Size of V texture buffer
 + OSRoundUp32B (THP video data horizontal size x vertical size/4)

// Size of buffer for THP audio data
+ OSRoundUp32B (THP audio data's maximum number of samples x 4) x  AUDIO_BUFFER_NUM

 // Size of work region used by THP library's low level API functions
 + THP_WORK_SIZE;
```

Formula 1 Size of the work region for the THP Simple Player

Note: The THPConv tool, which is used to create THP movie data, automatically sets the THP frame data's maximum size when it is creating THP movie data. This maximum size differs for every set of THP movie data. In addition, for movies that play back a long time or have frequently changing screens, this value will differ largely from the average size that can be calculated from the THP movie data file size. For example, in the sample data that comes with the THP library (rebirth.thp(dvddata/thpdemo/rebirth.thp), the average size is 39,552 bytes, and the maximum size is set to 77,728 bytes. This type of localized fluctuation in data size can have a large influence not only on the required buffer size for the Simple Player, but also on the data transfer speed from the optical disc and the CPU load required for decoding. You need to work to have a good balance when creating movies.

Note: The THPConv tool also automatically sets the maximum size for the THP audio data. Unlike for the video value, this value for the THP audio data does not fluctuate largely in every frame.

Note: The values for READ_BUFFER_NUM and for AUDIO_BUFFER_NUM are defined in the THP Simple Player's header file THPSimple.h(build/demos/thpdemo/include/THPSimple.h).

```
#define READ_BUFFER_NUM  10
#define AUDIO_BUFFER_NUM 3
```

Code 2 Definition of READ_BUFFER_NUM / AUDIO_BUFFER_NUM in THP Simple Player

Note: The value of THP_WORK_SIZE is defined in the THP library's header file thp.h(include/dolphin/thp.h).

```
#define THP_WORK_SIZE 0x1000
```

Code 3 Definition of THP_WORK_SIZE in THP library

Line 42:

Sets aside a work region for the Simple Player.

Line 49:

Registers in the Simple Player the work region that was set aside in line 42.

DOL-06-0183-001-A1
Released: 3/28/02

Line 61:

Several frames worth of THP movie data are read from the optical disc before the THP movie data begins to play. The `THPSimplePreLoad` function calls the `DVDRead` function to read the READ_BUFFER_NUM number of frames of THP movie data. The method of playback for the THP movie data is set in the argument of the `THPSimplePreLoad` function. LOOP is specified to loop the THP movie data, and NONE is specified to play the THP movie data only once in "one shot." Both LOOP and NONE are defined in the header file of the Simple Player `THPSimple.h`(build/demos/thpdemo/include/THPSimple.h).

```
#define NONE 0x00
#define LOOP 0x01
```

Code 4 Definitions of the argument of the THPSimplePreLoad function

Line 66:

The **start** flag is set, in order to get the timing for the start of THP audio data playback (see lines 97-102).

Line 80:

Decodes the THP video data. The decoded data is stored in YUV texture format in a buffer maintained internally by the Simple Player. If audio data is interleaved in the THP movie data, then the audio data is also decoded.

Line 86:

Draws the THP video data that was decoded in line 80. For drawing, the `THPSimpleDrawCurrentFrame` function takes the YUV texture-format data decoded by the `THPSimpleDecode` function and pastes it onto polygons.

Lines 88-93:

Writes from the EFB to the XFB and waits for two retraces.

When using the Simple Player, the loop portion in which the various Simple Player API functions are called and the frame buffer is drawn (i.e., the portion between lines 70-155), must have the same frame rate as that of the THP movie data being played. For example, if the Simple Player is incorporated into a loop where objects are being drawn at a frame rate of 60 frames per second, then the frame rate for that object rendering must drop to 30 frames per second, which is the frame rate for playback of THP movie data.

Note: This sample program is for the playback of THP movie data at a frame rate of 29.97 frames per second.

Lines 97-102:

Allows start of playback of THP audio data. If start of playback is allowed by the `THPSimpleAudioStart` function, decoded THP audio data is sent to the Nintendo GameCube audio interface. When audio data plays back interleaved THP movie data, `THPSimpleAudioStart` needs to be called once immediately after the start of playback.

The best location for the call to this function is immediately after the first frame (line 86) of THP movie data drawn by the `THPSimpleDrawCurrentFrame` function is displayed on the screen (at or after line 93).

Lines 104-110

Stops playback of THP movie data and restarts playback of that data at the beginning. To do this, first the `THPSimpleAudioStop` function (line 107) is called to cancel permission to play the audio data. When the denial is received, decoded THP audio data will not be sent to the Nintendo GameCube audio interface. Next, the `THPSimpleLoadStop` function (line 108) is called to stop preloading of THP movie data and to initialize the THPSimple structure. Doing so returns the Simple Player to its initial status.

Lines 112-122:

Exits playback of THP movie data. First the `THPSimpleAudioStop` function (line 115) is called to stop playback of audio data. Next the `THPSimpleLoadStop` function (line 116) is called to return the Simple Player to its initial status. Then the `THPSimpleClose` function (line 118) is called to close the THP file.

Lines 124 - 134:

Uses AX to play back or stop MIDI file.

Lines 136 - 144:

Turns up the playback volume of THP audio data

Lines 146-154:

Turns down the playback volume of THP audio data.

Line 157:

Exits the Simple Player. The `THPSimpleQuit` function calls the `LCDisable` function to disable locked cache and to return the Simple Player's internal state to the status it was in before the `THPSimpleInit` function was called. To use the Simple Player again, the process must begin with the calling of the `THPSimpleInit` function. When using AX simultaneously, call the `THPPlayerQuit` function while sound output of AX is set to not produce any sound.

DOL-06-0183-001-A1
Released: 3/28/02

3. The THP Simple Player API

This section provides explanations about the THP Simple Player's API functions and constants.

```
#include "THPSimple.h"
```

Code 5 Header file of the THP Simple Player API

3.1 READ_BUFFER_NUM

```
#define READ_BUFFER_NUM  10
```

Code 6 READ_BUFFER_NUM

READ_BUFFER_NUM is maintained inside the Simple Player and indicates the size of the buffer used for reading from the optical disc. The actual size of the buffer that is secured is equal to the maximum frame size kept in the THP movie data header (THPHeader), multiplied by READ_BUFFER_NUM.

3.2 AUDIO_BUFFER_NUM

```
#define AUDIO_BUFFER_NUM 3
```

Code 7 AUDIO_BUFFER_NUM

AUDIO_BUFFER_NUM is maintained inside the Simple Player, and indicates the size of the audio buffer. The actual size of the buffer that is secured is equal to the maximum sample number (x 4), kept in the header portion of the THP movie data (THPHeader), multiplied by AUDIO_BUFFER_NUM.

3.3 NONE / LOOP

```
#define NONE 0x00
#define LOOP 0x01
```

Code 8 NONE / LOOP

NONE and LOOP are used when the method of THP movie data playback is specified in the `THPSimplePreLoad` function. LOOP is used when the THP movie data is played in a loop; NONE is used when the THP movie data is played only once in one shot.

3.4 ALONE/WITH_AX/WITH_MUSYX

```
#define ALONE      0x00
#define WITH_AX    0x01
#define WITH_MUSYX 0x02
```

Code 9 ALONE/WITH_AX/WITH_MUSYX

Flags specified for the `THPSimpleInit` function. When Simple Player is used with AX simultaneously, specify **WITH_AX**. When used with MusyX simultaneously, specify **WITH_MUSYX**. When no audio library is used simultaneously, specify **ALONE**.

3.5 THPDecodeError

```
typedef enum
{
    DECODE_OK = 0,
    VIDEO_DECODE_FAIL,
    NO_READ_BUFFER,
    NO_AUDIO_BUFFER
} THPDecodeError;
```

Code 10 THPDecodeError

The various values defined by `THPDecodeError` are the error codes that are returned by the `THPSimpleDecode` function. The meaning of each error code is explained in Table 1 below.

Definition name	Code	Explanation of code
DECODE_OK	0	Function has terminated normally.
VIDEO_DECODE_FAIL	1	Decoding of THP video data has failed.
NO_READ_BUFFER	2	There is no data in the read buffer.
NO_AUDIO_BUFFER	3	There is no free audio buffer.

Table 1 Error codes returned by the `THPSimpleDecode` function

3.6 THPVideoInfo

```
typedef struct
{
    u32 xSize;
    u32 ySize;
} THPVideoInfo;
```

Code 11 THPVideoInfo

The THPVideoInfo structure holds the THP video data information (the vertical & horizontal sizes). The developer can reference this structure to get information about the THP video data.

The members of the THPVideoInfo structure are set by the `THPSimpleGetVideoInfo` function.

3.7 THPAudioInfo

```
typedef struct
{
    u32 sndChannels;
    u32 sndFrequency;
    u32 sndNumSamples;
} THPAudioInfo;
```

Code 12 THPAudioInfo

The THPAudioInfo structure holds the THP audio data information (the number of channels, the playback frequency, and the total number of samples). The developer can reference this structure to get information about the THP audio data.

The members of the THPAudioInfo structure are set by the `THPSimpleGetAudioInfo` function.

3.8 THPSimpleInit

```
BOOL THPSimpleInit(s32 audioSystem);
```

Code 13 THPSimpleInit

The `THPSimpleInit` function initializes the Simple Player's control structure (the THPSimple structure), enables locked cache and calls the `THPInit` function, which is one of the THP library's low level API functions. The `THPSimpleInit` function must be called before any of the other Simple Player functions. It also registers a callback in the Nintendo GameCube audio interface to play back THP audio data.

If using the Simple Player simultaneously with an audio library, be sure to call that audio library's initialization functions (`AXInit, sndInit`) before you call this function. If you are making simultaneous use of the AX library, be sure to call this function with AX sound output set to not produce any sound.

The audio library used simultaneously is specified as an argument of the `THPSimpleInit` function. When used with AX simultaneously, specify **WITH_AX**. When used with MusyX simultaneously, specify **WITH_MUSYX**. When no audio library is used simultaneously, specify **ALONE**.

This function returns TRUE if it has succeeded or FALSE if it has failed.

3.9 THPSimpleQuit

```
void THPSimpleQuit(void);
```

Code 14 THPSimpleQuit

The `THPSimpleQuit` function disables locked cache and returns the internal state of the Simple Player to the state it was in, before the `THPSimpleInit` function was called. To use the Simple Player again you must start again from the calling of the `THPSimpleInit` function. When Simple Player is used with AX, the `THPSimpleQuit` function needs to be called while sound output of AX is set to produce no sound.

The `THPSimpleQuit` function does not return a value.

3.10 THPSimpleOpen

```
BOOL THPSimpleOpen(char *fileName);
```

Code 15 THPSimpleOpen

The `THPSimpleOpen` function calls the `DVDOpen` function to open the THP movie data specified by the argument, and then calls the `DVDRead` function to read the header portion of that data. It then gets the necessary information from that header portion, and checks that the specified data is THP movie data that can be decoded.

The `THPSimpleOpen` function initializes the THP audio data playback volume to a value of 127.

The `THPSimpleOpen` function returns TRUE if it has succeeded or FALSE if it has failed.

3.11 THPSimpleClose

```
BOOL THPSimpleClose(void);
```

Code 16 THPSimpleClose

The `THPSimpleClose` function calls the `DVDClose` function to close the THP movie data that was opened by the `THPSimpleOpen` function.

Before the `THPSimpleClose` function is called it is necessary to first call `THPSimpleAudioStop` and `THPSimpleLoadStop` to halt all access to THP files by the Player.The `THPSimpleClose` function returns TRUE if it has succeeded or FALSE if it has failed.

DOL-06-0183-001-A1
Released: 3/28/02

3.12 THPSimpleCalcNeedMemory

```
u32 THPSimpleCalcNeedMemory(void);
```

Code 17 THPSimpleCalcNeedMemory

The `THPSimpleCalcNeedMemory` function calculates the size of work region needed by the Simple Player to playback the THP movie data.

The work region size differs depending on the THP movie data. You should call this function to obtain the correct size information, when starting to playback new THP movie data.

As the returned value, the `THPSimpleCalcNeedMemory` function returns the work region size if it has succeeded or 0 if it has failed.

3.13 THPSimpleSetBuffer

```
BOOL THPSimpleSetBuffer(u8 *buffer);
```

Code 18 THPSimpleSetBuffer

The `THPSimpleSetBuffer` function registers the work area from the argument in the Simple Player.

The work region size specified by the argument must be the same as the size returned by the `THPSimpleCalcNeedMemory` function.

The `THPSimpleSetBuffer` function returns TRUE if it has succeeded or FALSE if it has failed.

3.14 THPSimplePreLoad

```
BOOL THPSimplePreLoad(s32 loop);
```

Code 19 THPSimplePreLoad

The `THPSimplePreLoad` function prepares for the playback of THP movie data by calling the `DVDRead` function to read READ_BUFFER_NUM frame's worth of THP movie data.

The argument of the `THPSimplePreLoad` function sets the method of playback of THP movie data. The value of the argument is either LOOP or NONE. If LOOP is specified in the argument, the THP movie data is looped. If NONE is specified in the argument, then the movie data is played in one shot.

The `THPSimplePreLoad` function returns TRUE if it has succeeded or FALSE if it has failed.

3.15 THPSimpleAudioStart

```
void THPSimpleAudioStart(void);
```

Code 20 THPSimpleAudioStart

The `THPSimpleAudioStart` function allowss the passing of decoded THP audio data to the audio interface. This starts the playback of THP audio data.

The `THPSimpleAudioStart` function does not return a value.

3.16 THPSimpleAudioStop

```
void THPSimpleAudioStop(void);
```

Code 21 THPSimpleAudioStop

The `THPSimpleAudioStop` function prevents the passing of decoded THP audio data to the audio interface. This stops the playback of THP audio data.

The `THPSimpleAudioStop` function does not return a value.

3.17 THPSimpleLoadStop

```
BOOL THPSimpleLoadStop(void);
```

Code 22 THPSimpleLoadStop

The `THPSimpleLoadStop` function stops the preloading of movie data and initializes the Simple Player's control structure. This returns the Player to its initial state.

Once reading has stopped, the process must begin from the `THPSimplePreLoad` function, in order to begin playing the THP movie data again.

The Simple Player preloads THP movie data in the background, provided there is free space in the buffer it maintains for loading.

The `THPSimpleLoadStop` function returns TRUE if it has succeeded or FALSE if it has failed.

3.18 THPSimpleDecode

```
s32 THPSimpleDecode(void);
```

Code 23 THPSimpleDecode

The `THPSimpleDecode` function calls the THP library's low level API `THPVideoDecode` function and decodes THP movie data. If audio data is interleaved in the THP movie data, the same low-level API's `THPAudioDecode` function is called to decode the THP audio data. The decoded data is stored internally by the Player.

The `THPSimpleDecode` function returns the error codes shown in Table 1.

Note: Decoding of the THP audio data is performed by the Gekko CPU.

3.19 THPSimpleDrawCurrentFrame

```
s32 THPSimpleDrawCurrentFrame(GXRenderModeObj *rmode,
                              u32             x,
                              u32             y,
                              u32             polygonW,
                              u32             polygonH);
```

Code 24 THPSimpleDrawCurrentFrame

`THPSimpleDrawCurrentFrame` function draws by pasting THP video data decoded by the `THPSimpleDecode` function mentioned previously to the polygon with upper-left vertex at coordinate (x, y), and (width, height) = (polygonW, polygonH).

As the returned value, the `THPSimpleDrawCurrentFrame` function returns the frame number if it has succeeded, or -1 if it has failed.

DOL-06-0183-001-A1
Released: 3/28/02

3.20 THPSimpleSetVolume

```
BOOL THPSimpleSetVolume(s32 vol, s32 time);
```

Code 25 THPSimpleSetVolume

The `THPSimpleSetVolume` function sets the volume for playback of THP audio data. The Simple Player will change the playback volume of the THP audio data to the value specified by **vol** in the time specified by **time**. The **vol** argument has a range of values from 0 to 127. The **time** argument is set in units of milliseconds and has a range of 0 to 60000.

Note that the volume is initialized to a value of 127 when the `THPSimpleOpen` function is called.

For a return value, `THPSimpleSetVolume` function returns TRUE if volume setting has succeeded or FALSE if it has failed.

3.21 THPSimpleGetVolume

```
s32 THPSimpleGetVolume(void);
```

Code 26 THPSimpleGetVolume

The `THPSimpleGetVolume` function obtains the current volume set for playback of THP movie data.

For a return value, `THPSimpleGetVolume` function returns current playback volume if it has succeeded or –1 if it has failed.

3.22 THPSimpleGetVideoInfo

```
BOOL THPSimpleGetVideoInfo(THPVideoInfo *videoInfo);
```

Code 27 THPSimpleGetVideoInfo

The `THPSimpleGetVideoInfo` function gets the video information from the THP movie data that has been opened by the `THPSimpleOpen` function, and stores it in the THPVideoInfo structure specified by the argument.

The function returns TRUE if it has succeeded or FALSE if it has failed.

3.23 THPSimpleGetAudioInfo

```
BOOL THPSimpleGetAudioInfo(THPAudioInfo *audioInfo);
```

Code 28 THPSimpleGetAudioInfo

The `THPSimpleGetAudioInfo` function gets the audio information from the THP movie data that has been opened by the `THPSimpleOpen` function and stores it in the THPAudioInfo structure specified by the argument.

The function returns TRUE if it has succeeded or FALSE if it has failed.

3.24 THPSimpleGetFrameRate

```
f32  THPSimpleGetFrameRate(void);
```

Code 29 THPSimpleGetFrameRate

The `THPSimpleGetFrameRate` function gets the frame rate of the THP movie data that has been opened by the `THPSimpleOpen` function.

As the returned value, the `THPSimpleGetFrameRate` returns the frame rate if it has succeeded or `0.0f` if it has failed.

3.25 THPSimpleGetTotalFrame

```
u32  THPSimpleGetTotalFrame(void);
```

Code 30 THPSimpleGetTotalFrame

The `THPSimpleGetTotalFrame` function gets the total number of frames contained within the THP movie data that has been opened by the `THPSimpleOpen` function.

As the returned value, the function returns the frame total if it has succeeded or 0 if it has failed.

DOL-06-0183-001-A1
Released: 3/28/02

4. The THP Player API

Sections 2 and 3 explained the THP Simple Player (THPSimple, build/demos/thpdemp/src/THPSimple), with the goal of providing a basic understanding of the use of the THP library.

This section explains the API functions and constants of the THP Player (THPPlayer, build/demos/thpdemo/src/THPPlayer), created for use in applications.

```
#include "THPPlayer.h"
```

Code 31 Header file of the THP Player API

4.1 READ_THREAD_PRIORITY / AUDIO_THREAD_PRIORITY / VIDEO_THREAD_PRIORITY

```
#define READ_THREAD_PRIORITY  8
#define AUDIO_THREAD_PRIORITY 12
#define VIDEO_THREAD_PRIORITY 20
```

Code 32 READ_THREAD_PRIORITY / AUDIO_THREAD_PRIORITY / VIDEO_THREAD_PRIORITY

A maximum of three threads are created with the THP Player: a Read Thread for reading data from the optical disk, a Video Decode Thread for decoding the THP video data, and an Audio Decode Thread for decoding the THP audio data.

The order of priority of the various threads is indicated by READ_THREAD_PRIORITY / AUDIO_THREAD_PRIORITY / VIDEO_THREAD_PRIORITY. The priority of each thread is defined with the sample program, so you need to adjust it according to the particular environment being used if you are going to include the THP Player in a game. However, when you make changes, you need to maintain the following priorities for each thread.

```
High READ_THREAD_PRIORITY > AUDIO_THREAD_PRIORITY > VIDEO_THREAD_PRIORITY Low
```

Formula 2 Priority of each thread for THP Player

In order to read data smoothly from the optical disc, without any interruption in audio, the Read Thread and Audio Decode Thread must be assigned a high priority.

4.2 READ_BUFFER_NUM

```
#define READ_BUFFER_NUM  10
```

Code 33 READ_BUFFER_NUM

READ_BUFFER_NUM indicates the size of the buffer maintained inside the THP Player for reading data from the optical disc. The actual size of the buffer set aside, is equal to READ_BUFFER_NUM multiplied by the maximum frame size maintained in the THP movie data's header (THPHeader). Slight processing errors, that are generated due to the data loading speed, can sometimes be resolved by making READ_BUFFER_NUM larger.

4.3 DECODE_BUFFER_NUM

```
#define DECODE_BUFFER_NUM 3
```

Code 34 DECODE_BUFFER_NUM

DECODE_BUFFER_NUM indicates the size of the buffer maintained inside the THP Player to hold the decoded THP video data and the decoded THP audio data. The actual size of the buffer is calculated using the formula shown below:

```
// Size of Y texture buffer
( OSRoundUp32B (THP video data horizontal size x vertical size)
// Size of U texture buffer
+ OSRoundUp32B (THP video data horizontal size x vertical size / 4)
// Size of V texture buffer
+ OSRoundUp32B (THP video data horizontal size x vertical size / 4)
// Size of buffer for reading the THP audio data
+ OSRoundUp32B (THP audio data's maximum number of samples x 4) x DECODE_BUFFER_NUM
```

Formula 3 Formula for calculating size of set aside buffer

Slight processing errors that are generated, due to the duration of the decoding process, can sometimes be resolved by making DECODE_BUFFER_NUM larger.

4.4 STOP / PREPARE / PLAY / PLAYED / PAUSE / ERROR

```
#define STOP    0x0
#define PREPARE 0x1
#define PLAY    0x2
#define PLAYED  0x3
#define PAUSE   0x4
#define ERROR   0x5
```

Code 35 STOP / PREPARE / PLAY / PLAYED / PAUSE / ERROR

These six status values indicate the present status of the THP Player.

Status	Value	Explanation
STOP	0	The movie is stopped.
PREPARE	1	Preparations for movie playback are completed.
PLAY	2	The movie is playing.
PLAYED	3	The movie has played to the end. (For 'one-shot' playback, this indicates playback has reached end)
PAUSE	4	The movie is paused.
ERROR	5	An error has been generated.

Table 2 The status of the THP Player

DOL-06-0183-001-A1
Released: 3/28/02

4.5 NONE / LOOP / ODD_INT / EVEN_INT

```
#define NONE     0x0000
#define LOOP     0x0001
#define ODD_INT  0x0002
#define EVEN_INT 0x0004
```

Code 36 NONE / LOOP / ODD_INT / EVEN_INT

This is the playback flag specified in the argument for `THPPlayerPrepare`.

When you 'one-shot' playback THP movie data, specify **NONE**. When you loop playback, specify **LOOP**, and when THP movie data is an interlaced movie, specify **ODD_INT** or **EVEN_INT** depending on the starting field.

You can also specify the OR value of the above flag for `THPPlayerPrepare`. However, you must not specify **ODD_INT** and **EVEN_INT** at the same time.

4.6 ALONE/WITH_AX/WITH_MUSYX

```
#define ALONE      0x00
#define WITH_AX    0x01
#define WITH_MUSYX 0x02
```

Code 37 ALONE/WITH_AX/WITH_MUSYX

This is the flag specified in the argument for `THPPlayerInit`. When THP player is used with AX simultaneously, specify **WITH_AX**. When used with MusyX simultaneously, specify **WITH_MUSYX**. When no audio library is used simultaneously, specify **ALONE**.

4.7 THPVideoInfo

```
typedef struct
{
    u32 xSize;
    u32 ySize;
} THPVideoInfo;
```

Code 38 THPVideoInfo

The THPVideoInfo structure holds the information for the THP video data (the horizontal and vertical sizes). The developer can obtain THP video data information by referencing this structure.

The THPVideoInfo structure members are set by the `THPPlayerGetVideoInfo` function.

4.8 THPAudioInfo

```
typedef struct
{
    u32 sndChannels;
    u32 sndFrequency;
    u32 sndNumSamples;
} THPAudioInfo;
```

Code 39 THPAudioInfo

The THPAudioInfo structure holds the information for the THP audio data (number of channels, playback frequency and total number of samples). The developer can obtain THP audio data information by referencing this structure.

The THPAudioInfo structure members are set by the `THPPlayerGetAudioInfo` function.

4.9 THPPlayerInit

```
BOOL THPPlayerInit(s32 audioSystem);
```

Code 40 THPPlayerInit

The `THPPlayerInit` function initializes the Player's control structure (the THPPlayer structure), enables locked cache and calls the `THPInit` function, which is one of the THP library's low-level API functions. It also registers a callback function in the Nintendo GameCube audio interface for the playback of THP audio data. This function must be called before any of the other Player functions. If using the Player simultaneously with an audio library, be sure to call that audio library's initialization functions (`AXInit`, `sndInit`) before you call this function.

If you are making simultaneous use of the AX library, be sure to call the `THPPlayerInit` function with AX sound output set not to produce any sound.

The audio library used simultaneously is specified as an argument of `THPPlayerInit`. When used with AX simultaneously, specify **WITH_AX**. When used with MusyX simultaneously, specify **WITH_MUSYX**. When no audio library is used simultaneously, specify **ALONE.**

As the returned value, the `THPPlayerInit` function returns TRUE if the call has succeeded or FALSE if it has failed.

4.10 THPPlayerQuit

```
BOOL THPPlayerQuit(void);
```

Code 41 THPPlayerQuit

The `THPPlayerQuit` function disables locked cache and returns the internal status of the Player to the status it was in before the `THPPlayerInit` function was called. To use the Player again you must start again from the calling of the `THPPlayerInit` function.

If you are making simultaneous use of the AX library with the THP Player, be sure to call the `THPPlayerQuit` function with AX sound output set not to produce any sound.

The function does not return a value.

4.11 THPPlayerOpen

```
BOOL THPPlayerOpen(char *fileName, BOOL onMemory)
```

Code 42 THPPlayerOpen

The `THPPlayerOpen` function calls the `DVDOpen` function to open the THP movie data specified by the argument, and then calls the `DVDRead` function to read the header portion of that data. It then obtains the necessary information from that header portion and checks that the specified data is THP movie data that can be decoded. In the second argument, specify **FALSE** for streaming playback from the Nintendo GameCube Game Disc, or **TRUE** for On Memory playback.

The `THPPlayerOpen` function initializes the THP audio data playback volume to a value of 127.

As the returned value, the `THPPlayerOpen` function returns TRUE if the call has succeeded or FALSE if it has failed.

DOL-06-0183-001-A1
Released: 3/28/02

4.12 THPPlayerClose

```
BOOL THPPlayerClose(void)
```

Code 43 THPPlayerClose

The `THPPlayerClose` function calls the `DVDClose` function to close the THP movie data that was opened by the `THPPlayerOpen` function. Before calling the `THPPlayerClose` function, be sure to call the `THPPlayerStop` function to halt the playback of THP movie data.

As the returned value, the `THPPlayerClose` function returns TRUE if the call has succeeded or FALSE if it has failed.

4.13 THPPlayerCalcNeedMemory

```
u32 THPPlayerCalcNeedMemory(void)
```

Code 44 THPPlayerCalcNeedMemory

The `THPPlayerCalcNeedMemory` function calculates the number of bytes of work region needed to playback the THP movie data. The work region size differs, depending on the THP movie data. You should call this function to obtain the correct size information when starting to playback new THP movie data.

As the returned value, the `THPPlayerCalcNeedMemory` function returns the work region size if it has succeeded or 0 if it has failed.

4.14 THPPlayerSetBuffer

```
BOOL THPPlayerSetBuffer(u8 *buffer);
```

Code 45 THPPlayerSetBuffer

The `THPPlayerSetBuffer` function takes the work region required for playback of the THP movie data, specified in the argument, and sets it in the Player's structure. The size of the work region specified by the argument must be the same as the size returned by the `THPPlayerCalcNeedMemory` function.

As the returned value, the `THPPlayerSetBuffer` function returns TRUE if the call has succeeded or FALSE if it has failed.

4.15 THPPlayerPrepare

```
BOOL THPPlayerPrepare(s32 frameNum, s32 playFlag);
```

Code 46 THPPlayerPrepare

The `THPPlayerPrepare` function makes preparations for playback of the THP movie data. Depending on whether or not there is any THP audio data and on the form of playback (streaming or On Memory), the function creates a Video Decode thread, an Audio Decode thread and a Read thread. It then preloads the data and decodes the first frame. In addition, it registers a VI post-callback for control over playback (the registered VI post-callback can be called internally).

The first argument of the `THPPlayerPrepare` function specifies the frame number from which to start the THP movie data. If the THP movie data does not hold offset data for each frame, then 0 is the only value that can be set in this argument.

The playback flag can be specified via the second argument of the `THPPlayerPrepare` function. See section 4.5 "NONE / LOOP / ODD_INT / EVEN_INT" to learn which flags can be set. Multiple flags can be set using OR, but do not set both ODD_INT and EVEN_INT at the same time.

As the returned value, the `THPPlayerPrepare` function returns TRUE if the call has succeeded or FALSE if it has failed.

4.16 THPPlayerPlay

```
BOOL THPPlayerPlay(void)
```

Code 47 THPPlayerPlay

The `THPPlayerPlay` function initiates playback of THP movie data.

As the returned value, the function returns TRUE if the call has succeeded or FALSE if it has failed.

4.17 THPPlayerStop

```
BOOL THPPlayerStop(void);
```

Code 48 THPPlayerStop

The `THPPlayerStop` function stops playback of THP movie data. It stops the Video Decode thread, Audio Decode thread and Read thread that were created by the `THPPlayerPrepare` function and returns the VI post callback to its original status.

The `THPPlayerStop` function returns TRUE if the call has succeeded or FALSE if it has failed.

4.18 THPPlayerPause

```
BOOL THPPlayerPause(void);
```

Code 49 THPPlayerPause

The `THPPlayerPause` function temporarily pauses playback of THP movie data.

The function returns TRUE if the call has succeeded or FALSE if it has failed.

4.19 THPPlayerSkip

```
BOOL THPPlayerSkip(void);
```

Code 50 THPPlayerSkip

The `THPPlayerSkip` function advances the playback of THP movie data one frame forward.

The function returns TRUE if the call has succeeded or FALSE if it has failed.

DOL-06-0183-001-A1
Released: 3/28/02

4.20 THPPlayerDrawCurrentFrame

```
s32 THPPlayerDrawCurrentFrame(GXRenderModeObj *rmode,
                              u32              x,
                              u32              y,
                              u32              polygonW,
                              u32              polygonH);
```

Code 51 THPPlayerDrawCurrentFrame

THPPlayerDrawCurrentFrame function draws by pasting the THP video data that should be displayed at present to the polygon with upper-left vertex at coordinate (x, y), and (width, height) = (polygonW, polygonH).

As the returned value, the THPPlayerDrawCurrentFrame function returns the frame number if the frame could be drawn or -1 if the frame could not be drawn.

4.21 THPPlayerSetVolume

```
BOOL THPPlayerSetVolume(s32 vol, s32 time);
```

Code 52 THPPlayerSetVolume

THPPlayerSetVolume function sets playback volume for THP audio data. It changes from the current volume to the volume value specified by **vol** in the time specified by **time**. The value range you can specify for **vol** is from 0 to 127. **time** is specified in milliseconds. The value range you can specify for time is from 0 to 60000.

Be aware that when THPPlayerOpen function is called, volume value is initialized at 127.

For a return value, THPPlayerSetVolume returns TRUE if volume setting has succeeded, or FALSE if it has failed.

4.22 THPPlayerGetVolume

```
s32 THPPlayerGetVolume(void);
```

Code 53 THPPlayerGetVolume

The THPPlayerGetVolume function obtains playback volume for the current THP audio data.

For a return value, THPPlayerGetVolume returns the current playback volume if it has succeeded or –1 if it has failed.

4.23 THPPlayerGetVideoInfo

```
BOOL THPPlayerGetVideoInfo(THPVideoInfo *videoInfo);
```

Code 54 THPPlayerGetVideoInfo

The THPPlayerGetVideoInfo function obtains the video information of the THP movie data and stores it in the THPVideoInfo structure specified by the argument.

The function returns TRUE if the call has succeeded or FALSE if it has failed.

4.24 THPPlayerGetAudioInfo

```
BOOL THPPlayerGetAudioInfo(THPAudioInfo *audioInfo);
```

Code 55 THPPlayerGetAudioInfo

The `THPPlayerGetAudioInfo` function obtains the audio information of the THP movie data and stores it in the THPAudioInfo structure specified by the argument.

The function returns TRUE if the call has succeeded or FALSE if it has failed.

4.25 THPPlayerGetFrameRate

```
f32 THPPlayerGetFrameRate(void);
```

Code 56 THPPlayerGetFrameRate

The `THPPlayerGetFrameRate` function gets the frame rate of the THP movie data.

As the returned value, the function returns the frame rate if it has succeeded or `0.0f` if it has failed.

4.26 THPPlayerGetTotalFrame

```
u32 THPPlayerGetTotalFrame(void);
```

Code 57 THPPlayerGetTotalFrame

The `THPPlayerGetTotalFrame` function gets the total number of frames included in the THP movie data.

As the returned value, the function returns the total number of frames if it has succeeded or 0 if it has failed.

4.27 THPPlayerGetState

```
s32 THPPlayerGetState(void);
```

Code 58 THPPlayerGetState

The `THPPlayerGetState` function gets the current status of the Player. See section 4.4 “STOP / PREPARE / PLAY / PLAYED / ERROR” to read about the different status values.

As the returned value, this function returns the Player status. If the ERROR status comes back, you should terminate playback by calling the `THPPlayerStop` function and the `THPPlayerClose` function.

DOL-06-0183-001-A1
Released: 3/28/02

4.28 THPPlayerDrawDone

```
s32 THPPlayerDrawDone(void);
```

Code 59 THPPlayerDrawDone

`THPPlayerDrawDone` is called in lieu of `GXDrawDone`, in order to synchronize with the graphics processor at the time of movie playback. Internally, this function releases used THP video data, if any exists, after `GXDrawDone` has been called. This procedure guarantees that absolutely none of the used THP video data is accessed from the graphics processor and is safely released. If this function is not called, then the used THP video data will not be released and the Video Decode Thread will stop. Be sure to call `THPPlayerDrawDone` instead of `GXDrawDone`.

The `THPPlayerDrawDone` function does not return a value.

5. The THP Library's Low Level API

This section explains the THP library's low-level API functions and constants.

```
#include <dolphin/thp.h>
```

Code 60 The THP low level API header file

5.1 THP_WORK_SIZE

```
#define THP_WORK_SIZE              0x1000
```

Code 61 THP_WORK_SIZE

THP_WORK_SIZE indicates the work region size needed for decoding of THP video data.

5.2 Error codes

The `THPVideoDecode` function returns the following error codes:

Definition	Code	Explanation
THP_OK	0	The function has terminated normally.
THP_BAD_SYNTAX	3	THP video data has an invalid specification for the maker.
THP_UNSUPPORTED_QUANTIZATION	9	THP video data has an invalid specification for the number of quantization tables.
THP_UNSUPPORTED_PRECISION	10	THP video data has an invalid specification for the bit precision.
THP_UNSUPPORTED_MARKER	11	Unsupported marker inside THP video data.
THP_UNSUPPORTED_NUM_COMP	12	THP video data has an invalid specification for the number of components.
THP_UNSUPPORTED_NUM_HUFF	13	THP video data has an invalid specification for the number of Huffman tables.
THP_BAD_SCAN_HEADER	14	THP video data has an invalid scan header.
THP_INVALID_HUFFTAB	15	THP video data has an invalid specification for the Huffman table.
THP_UNSUPPORTED_COMPS	19	THP video data has an invalid specification for the components.
THP_NO_INPUT_FILE	25	Input data (THP video data) is not specified.
THP_NO_WORK_AREA	26	Work region is not specified.
THP_NO_OUTPUT_BUFFER	27	Output for decoded data is not specified.
THP_LC_NOT_ENABLED	28	Locked cache is not enabled.
THP_NOT_INITIALIZED	29	THPInit has not been called.

Table 3 THPVideoDecode function's error codes

5.3 THP_AUDIO_INTERLEAVE / THP_AUDIO_NO_INTERLEAVE

```
#define THP_AUDIO_INTERLEAVE    0x00
#define THP_AUDIO_NO_INTERLEAVE 0x01
```

Code 62 THP_AUDIO_INTERLEAVE / THP_AUDIO_NO_INTERLEAVE

THP_AUDIO_XXX is used for specifying the output format of the `THPAudioDecode` function.

THP_AUDIO_INTERLEAVE specifies that the `THPAudioDecode` function interleave the decoded left/right channel data in units of samples and output the data to the specified buffer. The decoded data is interleaved in the order (right, left, right, left...) as per the specifications of the Nintendo GameCube audio interface.

THP_AUDIO_NO_INTERLEAVE specifies that the `THPAudioDecode` function collect the decoded left/right channel data in each channel separately and output the data to the specified buffer. In this case, the decoded data is output in the order: (all samples of the right channel, followed by all samples in the left channel).

5.4 THPInit

```
BOOL THPInit(void);
```

Code 63 THPInit

The `THPInit` function initializes work region settings. It returns TRUE if call has succeeded or FALSE if it has failed.

Call the `THPInit` function before calling other THP functions. It is not a problem if you call `THPInit` more than once.

5.5 THPVideoDecode

```
s32 THPVideoDecode(void *file, void *tileY, void *tileU, void *tileV, void *work);
```

Code 64 THPVideoDecode

The `THPVideoDecode` function decodes the THP video data specified by the first argument, then writes the Y, U and V components of the decoded data to the buffers specified by the second, third and fourth arguments, respectively. The fifth argument specifies the work region used by the `THPVideoDecode` function. The size of work region is defined as THP_WORK_SIZE in the header file of the THP library, `thp.h(include/dolphin/thp.h)`.Locked cache should be put in the enabled state when calling this function.

The `THPVideoDecode` function returns the error codes shown in Table 3. These error codes can be broadly divided into three categories: **THP_OK** when the function terminates normally, **THP_NO_INPUT_FILE**, etc. when the function is used incorrectly, and **THP_BAD_SYNTAX**, etc. when the THP video data cannot be decoded. If the function has been used incorrectly, correct the error and call the `THPVideoDecode` function again. If the THP video data cannot be decoded, check to see that the first argument specifies the correct address, because it may be that the data specified by the first argument is not THP movie data.

The THPVideoDecode function outputs the three components of the decoded data in 8-bit texture image format. The size of the buffer for each component specified in the function is calculated as shown in Formula 4 below. The starting address of each buffer must be 32byte aligned.

Component	Size (Bytes)
Y	Number of horizontal pixels x Number of vertical pixels
U	Number of horizontal pixels x Number of vertical pixels / 4
V	Number of horizontal pixels x Number of vertical pixels / 4

Formula 4 Size of buffer specified in THPVideoDecode function for each component

Note: The THPViodeDecode function utilizes locked cache when decoding THP video data. The function's computational results could get destroyed if it is executing inside one thread when a thread with higher priority also makes use of locked cache. You thus need to handle the THPViodeDecode function carefully.

5.6 THPAudioDecode

```
u32 THPAudioDecode(s16 *buffer, u8 *audioFrame, s32 flag);
```

Code 65 THPAudioDecode

The THPAudioDecode function decodes the THP audio data specified by the second argument and outputs it to the buffer specified by the first argument. The third argument specifies the format for the data being output. The THPAudioDecode function outputs the audio data in stereo. If the function receives monaural THP audio data, it will convert it internally into stereo.

As the returned value, the THPAudioDecode function returns the number of decoded stereo samples (the number increases by 1 for each pair of left and right channel sample).

The flag of the third argument is specified as either THP_AUDIO_INTERLEAVE or THP_AUDIO_NO_INTERLEAVE.

THP_AUDIO_INTERLEAVE specifies that the THPAudioDecode function interleave the decoded left and right channel data in units of samples and output the data to the specified buffer. The decoded data is interleaved in the order (right, left, right, left...), as per the specifications of the Nintendo GameCube audio interface.

THP_AUDIO_NO_INTERLEAVE specifies that the THPAudioDecode function collect the decoded left/right channel data in each channel separately and output the data to the specified buffer. In this case, the decoded data is output in the order: (all samples of the right channel, followed by all samples in the left channel).

Note: If the THP audio data is monaural, the THPAudioDecode function will convert the decoded data into stereo no matter what is specified in the third argument. Set the size of the buffer specified in the first argument the same as you would for stereo THP audio data.

Note: The THPAudioDecode function does not utilize locked cache.

-

6. How to Create THP Movie Data

6.1 Overview

The format of THP movie data has specifications unique to the Nintendo GameCube. THP movie data is composed of THP video data and THP audio data, and both types of data are interleaved in every frame. THP movie data has high extensibility, and can also incorporate a third type of data besides THP video data and THP audio data.

A unique format is employed for the THP video data. It has been customized for rapid decoding by the Nintendo GameCube. The JPEG data is converted into THP video data very quickly, with no loss in picture quality from the original JPEG data.

The THP audio data is compressed in the Nintendo GameCube audio system's DSP ADPCM format. During conversion from the original 16-bit PCM data, the playback frequency is slightly altered (48,000Hz → 48,043Hz / 32,000Hz → 32,028Hz) to achieve synchronization with the THP video data. This THP audio data supports both monaural and stereo playback.

The utility THPConv is available for use in creating THP movie data. The THPConv tool supports these two input data formats: QuickTime Motion (Photo) JPEG files (see Section 6.2.1 "Converting QuickTime Motion JPEG files into THP movie data") and sequential JPEG files (see Section 6.2.2 "Converting sequential JPEG files into THP movie data"). The tool also has a function that can be used for replacing the audio data that is already incorporated in the THP movie data (see Section 6.2.3 "Replacing the THP audio data inside the THP movie data").

6.2 How to use

The THPConv tool is a Win32 console application. The tool is used differently depending on the format of the input data. The following sections explain how to use the THPConv tool for each separate input data format.

Addendum: The method of use can be displayed by executing the THPConv tool without any options.

6.2.1 Converting QuickTime Motion JPEG files into THP movie data

Use the THPConv tool as follows when converting QuickTime style Motion JPEG files to THP movie data.

```
  THPConv.exe -m <inputfile> -d <outputfile> [-<option> <argument>]
```

```
-m <inputfile>     Specifies the input file (a QuickTime Motion JPEG file).  This argument must
                   be specified.

-d <outputfile>    Specifies the output file (the THP file).  If this argument is not specified,
                   then the THPConv tool will automatically create a file that has the same name
                   as the input file, but with the extension changed to .thp.
```

The THPConv tool supports the following options:

-s <wavefile> Specifies an audio file (a wav file) to convert into THP audio data. The audio data specified with this argument has priority for conversion into THP audio data, and the audio data inside the Motion JPEG file is ignored.

-r <frame rate> Specifies the movie frame rate. The frame rate can be set in the range of 1.0 to 59.94. If no value is specified, then the default value (29.97) is utilized.

-o With THP movie data, the offset to each frame's data can be stored as a table (see Section 6.3.4.3). When this argument is specified, the THPConv tool creates an offset table inside the THP data. If this argument is not specified, then an offset table will not be created inside the THP data.

-v Turns the verbose mode on.

6.2.2 Converting sequential JPEG files into THP movie data

Use THPConv tool as follows when converting serial numbered JPEG files to THP movie data.

THPConv.exe -j <*.jpg> **-d** <outputfile> [-<option> <argument>]

-j <*.jpg> Specifies the input files (sequential JPEG files). Wildcard characters (*) can be used. This argument must be specified.

-d <outputfile> Specifies the output file (the THP file). This argument must be specified.

The THPConv tool supports the following options:

-s <wavefile> Specifies an audio file (a wav file) to convert into THP audio data. The audio data specified with this argument has priority for conversion into THP audio data, and the audio data inside the Motion JPEG file is ignored.

-r <frame rate> Specifies the movie frame rate. The frame rate can be set in the range of 1.0 to 59.94. If no value is specified, then the default value (29.97) is utilized.

-o With THP movie data, the offset to each frame's data can be stored as a table (see Section 6.3.4.3). When this argument is specified, the THPConv tool creates an offset table inside the THP data. If this argument is not specified, then an offset table will not be created inside the THP data.

-v Turns the verbose mode on.

DOL-06-0183-001-A1
Released: 3/28/02

In the example below, the THPConv tool is used to convert sequential JPEG files into THP movie data.

```
THPConv -j test*.jpg -p output.thp
```

When this is executed on the command line, the sequential JPEG files located in the current directory (test001.jpg, test002.jpg, test003.jpg ....) are individually converted into THP video data, and then collected together in one THP movie data file named `output.thp`. See section 6.3.2 "Sequential JPEG files" for information about sequential JPEG files.

6.2.3 Replacing the THP audio data inside the THP movie data

```
THPConv.exe -c <inputfile> -s <wavfile> -d <outputfile> [-<option> <argument>]
```

-c <inputfile> Specifies the input file (a THP file). This argument must be specified.

-s <wavfile> Specifies the sound file (a wav file) that will be substituted. This argument must be specified.

-d <outputfile> Specifies the output file (a THP file). If this argument is not specified, then the THPConv tool will **overwrite the input file.**

The THPConv tool supports the following options:

-r <frame rate> Specifies the movie frame rate. The frame rate can be set in the range of 1.0 to 59.94. If no value is specified, then the default value (29.97) is utilized.

-o With THP movie data, the offset to each frame's data can be stored as a table (see Section 6.3.4.3). When this argument is specified, the THPConv tool creates an offset table inside the THP data. If this argument is not specified, then an offset table will not be created inside the THP data.

-v Turns the verbose mode on.

Note: When the THPConv tool replaces the audio data that is already inside the THP movie data, the old THP audio data is deleted. Once the THP audio data has been deleted it cannot be recovered. Moreover, if an output file is not specified when the audio data is replaced, then the THPConv tool will overwrite the existing THP movie data. Thus, when replacing audio data, take the precaution of creating a backup file or specifying an output file, so you do not inadvertently lose data.

Note: If the playback duration of the wav file is longer than that of the THP movie data, then when the wav file is substituted for the existing audio data, that portion of the wav file that is longer than the THP movie data will not be converted. Conversely, if the playback duration of the THP movie data is longer than that of the wav file, then there will be no sound during that extra time.

Note: If the THP movie data that has been specified as the input file does not contain audio data, this operation will add THP audio data to the specified file.

DOL-06-0183-001-A1
Released: 3/28/02

6.3 Data Formats

6.3.1 QuickTime Motion JPEG Files

The THPConv tool can convert QuickTime Motion (Photo) JPEG files into THP movie data. However, the following restrictions apply to the video data and the audio data stored inside the Motion JPEG file:

- **The video data**
  - Only the JPEG baseline DCT format is supported
  - Only sequential coding is supported
  - Only 4:2:0 sub-sampling is supported
  - The pixel count must be a multiple of 16 both vertically and horizontally
- **The audio data**
  - Must be uncompressed 16-bit PCM data
  - Monaural (1 channel) and stereo (2 channels) are supported
  - Only playback frequencies of 32KHz and 48KHz are supported

If the Motion JPEG file does not fully meet these restrictions, then the THPConv tool will output an error and the process will stop.

Note: The THPConv tool only supports Photo-JPEG codec QuickTime Motion JPEG files. It does not support Motion JPEG A/B.

Note: When the THPConv tool converts the Motion JPEG file's audio data into THP audio data, the playback frequency is altered (48,000Hz → 48,043Hz / 32,000Hz → 32,028Hz) to match the characteristics of the Nintendo GameCube. It is not necessary to create a QuickTime file ahead of time, with the playback frequency already changed for the Nintendo GameCube. However, if the playback frequency of the audio data is something other than 48,000Hz or 32,000Hz, then the THPConv tool will output an error and the process will stop.

6.3.2 Sequential JPEG files

The THPConv tool can convert sequential JPEG files (a group of JPEG files, with consecutive numbers at the end of the files that indicate the ordered sequence for playback) into THP movie data. If audio data is interleaved in the sequential JPEG files, either a wav file must be specified with the -s option during the conversion process, or the wav file must be added after the sequential JPEG files have been converted.

The following restrictions apply to the sequential JPEG files that can be handled by the THPConv tool:

- Consecutive numbers corresponding to the frame numbers must be at the end of the sequential JPEG files.
- The JPEG file that corresponds to the first frame of the THP movie data can have any number, but the files thereafter must be numbered consecutively from that number.
- The frame number at the end of the filename should have the same number of digits as the final frame, with zeros attached to the front.
- All of the JPEG files that are collected together into a single set of THP movie data must have the same number of vertical pixels and the same number of horizontal pixels.

In the example below, sequential JPEG files are used to create 4 seconds worth of THP movie data at a frame rate of 30 frames per second. The sequential JPEG files (testxxxx.jpg) are numbered as shown below:

```
Frame     1 : test001.jpg
Frame     2 : test002.jpg
Frame     3 : test003.jpg
   :            :
Frame    51 : test051.jpg
Frame    52 : test052.jpg
   :            :
Frame   118 : test118.jpg
Frame   119 : test119.jpg
Frame   120 : test120.jpg
```

The following restrictions apply to the individual JPEG files that comprise the sequential JPEG file group:

- Only the JPEG baseline DCT format is supported
- Only sequential coding is supported
- Only 4:2:0 sub-sampling is supported
- The pixel count must be a multiple of 16 both vertically and horizontally

If the sequential JPEG files do not fully meet these restrictions, then the THPConv tool will output an error and the process will stop.

6.3.3 wav files

The THPConv tool can convert a common wav file into THP audio data, and interleave it into the THP movie data. However, there are certain restrictions as to what kinds of wav files the THPConv tool can convert:

- Must be uncompressed 16-bit PCM data
- Monaural (1 channel) and stereo (2 channels) are supported
- Only playback frequencies of 32KHz and 48KHz are supported

If the wav file does not fully meet these restrictions, then the THPConv tool will output an error and the process will stop.

Note: When the THPConv tool converts the wav file's audio data into THP audio data, the playback frequency is altered (48,000Hz → 48,043Hz / 32,000Hz → 32,028Hz), to match the characteristics of the Nintendo GameCube. It is not necessary to create a wav file ahead of time, with the playback frequency already changed for the Nintendo GameCube. However, if the playback frequency of the audio data is something other than 48,000Hz or 32,000Hz, the THPConv tool will output an error and the process will stop.

6.3.4 THP movie data

The following sections explain the format of the THP movie data that is output by the THPConv tool.

6.3.4.1 THPHeader

THPHeader is situated at the beginning of the THP movie data. It holds information for the proper initialization of the THP Player.

"THP\0" (magic[4]) and the version number are stored at the start of THPHeader, in order to identify the THP movie data. The upper two bytes of the version number indicate the major number and the lower 2 bytes the minor number. The THPConv tool automatically sets the version number. THP_VERSION is defined in the header file (THPSimple.h, THPPlayer.h) of both Players.

Following this information inside THPHeader, is information about the maximum frame data size (bufSize) and the maximum number of audio samples (audioMaxSamples), which together are used by the THP Player to calculate the size of the work region.

After this comes other information, including the THP movie data's frame rate (frameRate) and the total number of frames (numFrames).

The THP movie data format is designed with extensibility in mind, and can accommodate additional data. The offset to this additional data also can be stored in THPHeader (beginning from finalFrameDataOffsets).

6.3.4.2 THPFrameCompInfo

Data can be interleaved in frames and stored inside the THP movie data. In the THP library, these interleaved data are referred to as components. THP video data and THP audio data are also components. Designed with extensibility in mind, the THP movie data can store components other than video and audio.

THPFrameCompInfo holds numComponents, which is the number of components included in the THP movie data, and frameComp[], an array showing the order of the various components stored in each frame. This array stores the THP Components descriptors (see table below) in the order in which the components are interleaved.

Video Comp	0
Audio Comp	1
Undefined	2
Undefined	3
Undefined	4
Undefined	5
Undefined	6
Undefined	7
-	8
-	9
-	10
-	11
-	12
-	13
-	14
-	15
Nothing	FF

Table 4 THP Components descriptors

For example, if the first component of the frame is THP video data and the second component is THP audio data, then THPFrameCompInfo would look like this:

```
numComponents       = 2    : Number of components
frameComp[0]        = 0    : Video data descriptor
frameComp[1]        = 1    : Audio data descriptor
frameComp[2..15]    = FF   : No data
```

After THPFrameCompInfo comes information on each component (THPVideoInfo, THPAudioInfo, etc.). This latter component information must also be in the order of the THP Components descriptors stored in the frameComp array.

6.3.4.2.1 THPVideoInfo

THPVideoInfo holds the THP video data's vertical & horizontal size information (xSize, ySize). This information is used for proper playback of the THP video data.

6.3.4.2.2 THPAudioInfo

THPAudioInfo holds the number of channels (sndChannels), the playback frequency (sndFrequency) and the total number of audio samples (sndNumSamples) of THP audio data. This data is used for proper playback of the THP audio data.

6.3.4.3 THPFrameOffsetData

The THP movie data can also maintain a data table with the offset to the start of every frame. When the THPConv tool is used with the -o option, THPFrameOffsetData is created in the THP movie data. If the -o option is not specified, then THPFrameOffsetData will not be created. Use this offset data table for special playback purposes, such as to start movie playback from any frame.

6.3.4.4 MovieData

MovieData holds the interleaved component data for each frame.

6.3.4.4.1 FrameHeader

FrameHeader is situated at the front of the MovieData data for every frame.

Stored at the start of FrameHeader are the size of the previous frame (frameSizePrevious) and the size of the next frame (frameSizeNext). For the very first frame, the size of the very last frame is stored in frameSizePrevious. Similarly, for the very last frame, the size of the very first frame is stored in frameSizeNext.

Following these two values, FrameHeader stores the data size information in each component interleaved in that frame. This size information must be stored in the order in which the components are interleaved, as defined in THPFrameCompInfo.

The size of each frame must be a multiple of 32 bytes. Frames should be padded with 0, as needed, at the end of the frame data to meet this requirement.

Note: The THP audio data is compressed in the Nintendo GameCube audio system's DSP ADPCM format. Because of this, the number of audio data samples stored in every frame (with the exception of the last frame) must be a multiple of 14.

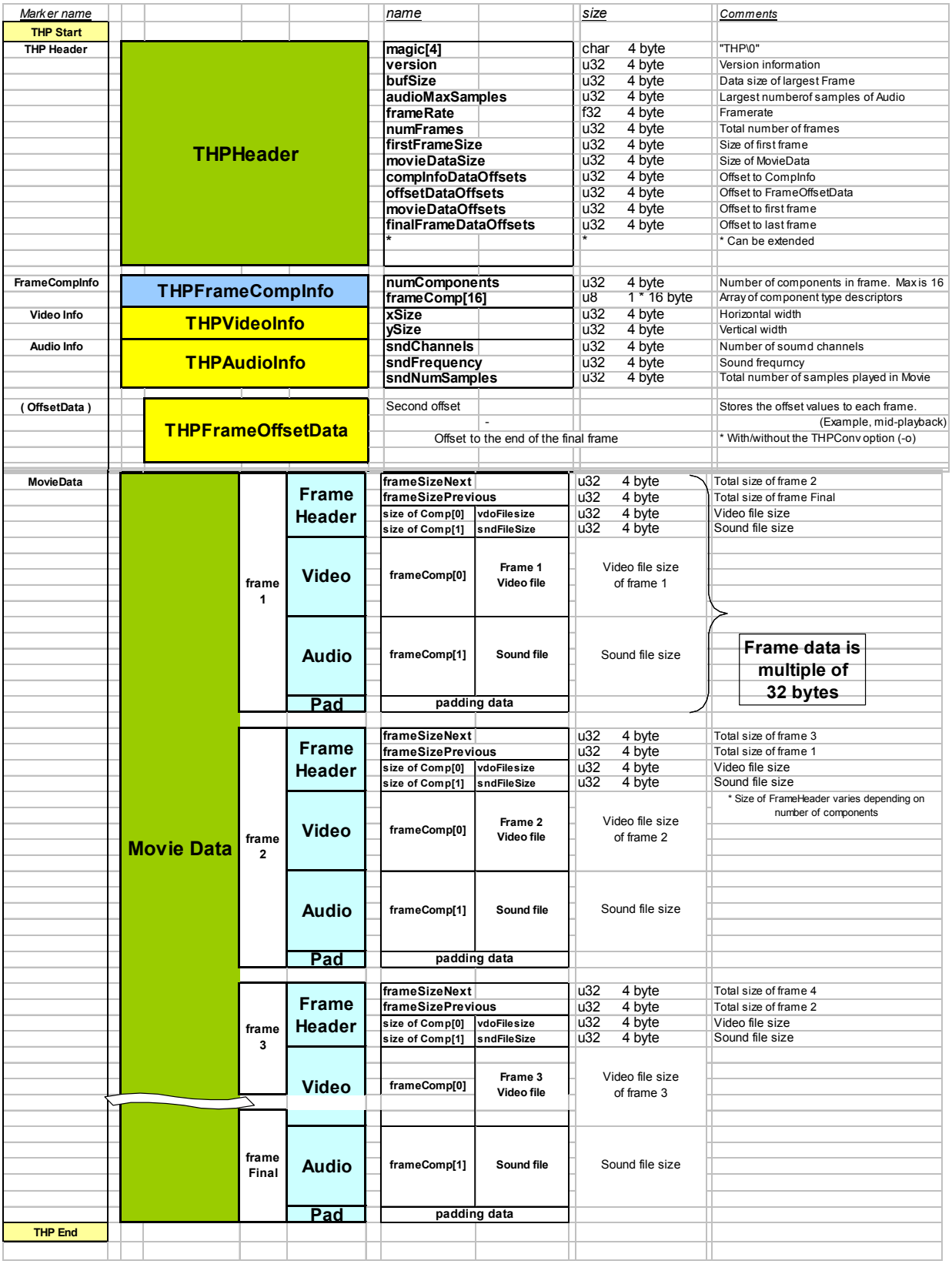

Figure 1 THP file format

6.4 Items to Note When Creating THP Movie Data

The following items should be noted when using the THPConv tool.

6.4.1 Caution regarding the path

The THPConv tool makes use of dsptool(D).dll, which comes with the Nintendo GameCube SDK. In order to use THPConv tool, you must adhere to the path: \(DolphinRoot)\x86\bin.

6.4.2 Sufficient free hard-disk space

The THP movie data output by the THPConv tool is nearly the same size as the original data before the conversion. When the THPConv tool converts video data and audio data into THP movie data, it creates temporary files in the current directory. These temporary files are also around the same size as the original data.

Be sure to confirm that there is sufficient free space on the hard disk when using the THPConv tool.

```
Necessary free hard-disk space = Size of original data x 3
```

Formula 5 Required free hard-disk space when using THPConv

When the THPConv tool creates temporary files, it names the temporary file for video data __tmp_VD and the temporary file for audio data __tmp_AD. If files with these same filenames exist in the current directory, the THPConv tool will overwrite them.

6.4.3 The execution environment for the THPConv tool

If the THPConv tool is executed on bash, sometimes the tool will not get the input file that was specified by the argument and the THP file will not be created as anticipated. For this reason, be sure to execute the THPConv tool from the command line.

7. Cautions

7.1 General Cautions

7.1.1 Sub-sampling

The THP library only supports 4:2:0 sub-sampling.

For a JPEG file with 4:2:0 sub-sampling, the Cb & Cr components are culled as shown in the figure below in regards to the Y component:

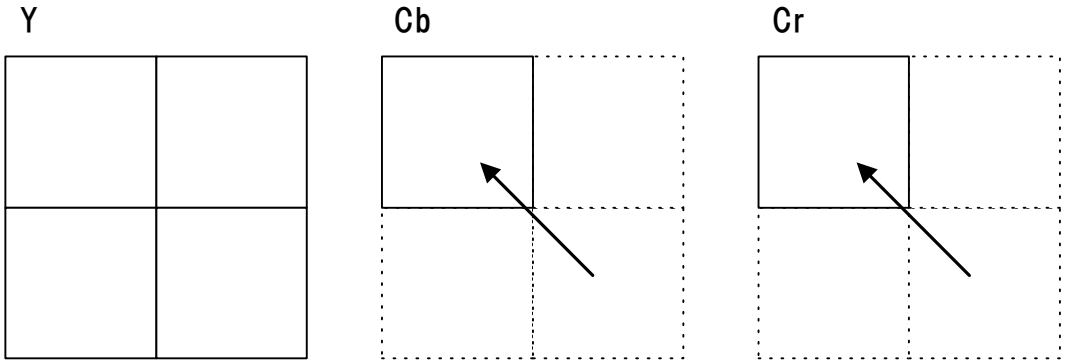

Figure 2 4:2:0 sub-sampling

If the input JPEG data does not have 4:2:0 sub-sampling, then the THPConv tool will output an error and the conversion process will stop. If it is not clear what kind of sub-sampling is performed on the JPEG data output by a commercial graphics application, you can determine whether the THP library will be able to decode the data by seeing what happens when the file is input to the THPConv tool.

7.1.2 The Players and the GX settings

The THP Simple Player and the THP Player both utilize the THP library's low-level API function `THPVideoDecode` to decode the THP video data. This function outputs the decoded data in YUV texture format. Both Players draw to the screen by pasting this YUV texture format decoded data to polygons.

Both Players effect major changes to the GX settings when drawing the decoded data. For applications that make use of THP movie data, and especially for applications that play movies and draw objects at the same time, be sure to rectify the GX settings that are used for the drawing of objects, fixing them in each frame after the movie has been drawn and before the object is drawn.

7.1.3 Output of THP Audio Data for Players

When the THP Simple Player and THP Player are used with AX or MusyX simultaneously, they mix audio output data from the various audio libraries with THP audio data using the CPU, and send it to the audio interface.

7.1.4 Sampling rate of THP audio data with simultaneous use of audio libraries

The sampling rate for THP audio data needs to be 32khz when the THP Simple Player and the THP Player are used with AX and MusyX simultaneously.

7.1.5 Monitoring the optical disc drive during streaming playback

When streaming THP movie data from the optical disc for playback, please monitor the state of the disc drive from inside the main loop that calls the THP Player's various API functions and draws the frame buffer, and perform the proper process for the given disc drive state. The code shown below is an example of such processes, and you do not need to follow the exact same procedure.

```
switch(DVDGetDriveStatus())
{
    case DVD_STATE_FATAL_ERROR:
        Stop playback
        break;
    case DVD_STATE_NO_DISK:
        Stop playback
        break;
    case DVD_STATE_COVER_OPEN:
        Pause playback
        break;
    case DVD_STATE_WRONG_DISK:
        Stop playback
        break;
    case DVD_STATE_RETRY:
        Stop playback
        break;
 }
```

Code 66 Example of monitoring drive inside main loop

7.1.6 Handling the sample data

The sample data that accompanies this library (rebirth.thp) is provided strictly for reference purposes. The misappropriation of this sample data and its duplication or alteration without the permission of Nintendo of America is prohibited.

* Sample data [rebirth.thp] : Copyright (c) 2000 Nintendo

Producer	mix core（http://www.mix-core.com）
CG Designers	Shunichi Shirai, Yutaka Nishikawa, Makoto Nishibori,
	Takahiro Onishi, Hiroyuki Kusugawa
Sound Composer	Masaya Tsunemoto
Violinist	Yoko Yoshida

* Please direct all queries regarding this sample data to Nintendo of America.

7.2 Cautions regarding the THP Player

7.2.1 The main loop

When playing THP movie data, make sure that the main loop that draws to the screen and calls the THP Player functions is created with the loop set to 1vsync.

```
THPPlayerPlay()

while(1)
{
            .
            .

    THPPlayerDrawCurrentFrame();
            .
            .

    VISetNextFrameBuffer();
    VIFlush();
    VIWaitForRetrace();
}
```

Code 67 Restriction on main loop when using the THP Player

The THP movie data may not play smoothly if the main loop is set to loop at anything other than 1vsync.

7.2.2 The THPPlayerDrawCurrentFrame function

Sometimes the call to the `THPPlayerDrawCurrentFrame` function may fail (returned value = -1) even after the call to the `THPPlayerPlay` function has succeeded. The reason is because, internally, playback has been delayed past the proper timing for the start of play.

Basically speaking, the first frame of decoded data is fetched by the THP Player's VI post callback function at the time of the vsync right after the `THPPlayerPlay` function is called (though in the case of an interlaced movie, this action could be delayed due to restrictions on the first field). It is after this process that drawing of video data by the `THPPlayerDrawCurrentFrame` function becomes possible.

If you are drawing movies and objects at the same time, please check the value returned by the `THPPlayerDrawCurrentFrame` function and confirm that it is some frame number other than -1 (failure) before displaying the object.

If it is not an interlaced movie, then you can use the procedure below to make certain that the `THPPlayerDrawCurrentFrame` function is called successfully:

```
THPPlayerPlay();

VIWaitForRetrace(); <------- At the time of this vsync, the first frame is
                             fetched by the VI post callback.
while(1)
{
       .
       .

    THPPlayerDrawCurrentFrame();
       .
       .

    VISetNextFrameBuffer();
    VIFlush();
    VIWaitForRetrace();
}
```

Code 68 How to make certain the call to the THPPlayerDrawCurrentFrame function succeeds

7.2.3 VI post callback

The THP Player utilizes VI post callbacks in order to control movie playback. Callbacks that are registered before the registration of the THP Player's VI post callback function are called via the Player's VI post callback. At the end of playback (i.e., when `THPPlayerStop` is called), the Player returns the VI post callback to its origin.

7.2.4 Interlaced movies

The THP Player supports the playback of interlaced movies, but only if the format is such that the data in every frame contains even fields and odd fields that are alternately interleaved in each scan. The even and odd fields should be set up so that each field comes 1/59.94 seconds after the previous field for NTSC and MPAL, and 1/25 seconds for PAL

When the data exists in this format, the timing for the start of playback is affected by whether the data in each frame begins with the even field or the odd field.

The THP Player uses the second argument of the `THPPlayerPrepare` function to get information on the scanning order of the interlaced movie, and to automatically adjust the timing of the start of playback.

There are no restrictions on the screen size for interlaced movies that can be played by the THP Player. However, whether each frame begins with an even field or an odd field places restrictions on the drawing location when the movie is drawn to the screen. The developers must adjust the drawing location themselves.

When an interlaced movie is scaled, the position of the even and odd lines inside the frame will be misaligned, and it may not be possible to play the movie back correctly. For this reason, do not scale interlaced movies for playback.

Below is a list of the restrictions that apply to interlaced movies that can be played with the THP Player:

- The even fields and the odd fields must be interleaved.
- The frame rate must be 29.97 frames per second for NTSC and MPAL, and 25 frames for PAL.
- The developer must specifiy in the THP Player whether the frame data that comprises the interlaced movie is scanned in the order of even field → odd field, or odd field → even field.
- There are no restrictions on screen size, but the drawing location must be adjusted.
- The decoded data canot be scaled.